N89-19818

THE SCHEDULING TECHNIQUES OF ESP2

John P. Jaap
and
Elizabeth K. Davis

Mission Integration/EL22
Marshall Space Flight Center
Huntsville, Al 35812

ABSTRACT

The Mission Analysis Division of the Systems Analysis and Integration Laboratory at the Marshall Space Flight Center has developed a robust automatic scheduler which can produce detailed schedules for the multi-step activities required for payload operations on the Space Station. This scheduler, a part of the Expert Scheduling Program (ESP2), has five components: the bookkeeper, checker, loader, selector, and explainer. The bookkeeper maintains the usage profiles for nondepletable resources, consumables, equipment, crew, and the times of all the steps for the payload activities for several different schedules simultaneously. The checker searches the data maintained by the bookkeeper and finds times when the constraints of each step of an activity are satisfied. The loader is an expert system which uses the techniques of forward chaining, depth-first searching, and backtracking to manage the workings of the checker so that activities are placed in the schedule without violating constraints (such as crew, resources, and orbit opportunities). The selector has several methods of choosing the next activity for the loader to schedule; new methods using rule-based technology are being studied. The explainer shows the user why an activity was or was not scheduled at a certain time; it offers a unique graphical explanation of how the expert system (the loader) works.

INTRODUCTION

The Experiment Scheduling Program (ESP) has been used to schedule the payload activities of most Spacelab missions and several partial payloads. For Space Station, features are being added to ESP that will automate many of the tasks that now must be done by an expert user. New strategies, rules, and capabilities for scheduling are being developed. A new name, the Expert Scheduling Program, and a new acronym, ESP2, have been adopted to emphasize the improvements being made to the program. These improvements are integrated into this presentation.

Before a description of the scheduling process is given, a few facts about the input data should be stated. The input database to ESP2 contains a mission model, multiple payload activity models each with multiple steps, and the orbit opportunity timelines required by the activity models. The paper entitled "Space Station Payload Operations Scheduling with ESP2" published concurrently with this paper describes the activity modeling capabilities. The program converts the activity requirements to feasibility tests, availability windows, and backtracking rules. The synthesis of the availability windows and backtracking rules into a schedule is the focus of this paper.

THE SCHEDULING PROCESS

When generating a trial schedule, ESP2 first initializes the schedule. Then it begins the "select, check, insert" loop which continues until all requested performances are attempted. The checking process is based on calculating windows which are nested in a hierarchy such that each lower window is totally contained within the window above it. At each level of the hierarchy, a window may contain many windows at the next lower level, and each of these may contain many windows. Checking of constraints begins at the topmost window and proceeds downward and across. When an acceptable window is found, checking proceeds immediately to the next lower level. If, at any level, an acceptable window cannot be found, the window above it is failed, and the next window at the level of the failed window is defined. This technique is a micro-example of depth-first searching. Checking is successfully completed whenever an acceptable window is found at the bottom level. Checking fails whenever another window at the top level cannot be defined. Models with two-way concurrency (mandatory concurrency) are processed simultaneously. There are eleven different types of windows within the hierarchy. Five of these apply to the entire performance and six apply to each step of the activity model. The windows for each step are independently nested; i.e., the crew subwindow for step 5 of a model is

nested within the nondepletable/equipment subwindow for step 5, but is not nested within any of the windows for the other steps on that model. If a model or step does not have a particular requirement, the associated window is set equal to the next higher level window.

The scheduler is described by presenting on a step-by-step basis how an expert mission planner would do the task manually and then by showing how this approach is implemented in ESP2. Since the scheduler emulates how an expert would do the task manually it meets the primal definition of an "expert system." When explaining the scheduling process, it is convenient to start at the bottom (the bookkeeper) and work upwards to the selector.

Figure 1 shows the major components of the program (only ESP) and their interfaces. While the primary purpose of the bookkeeper and the checker is to support the loader, they also support other features of ESP2 such as retrieving a previously generated schedule from a file, schedule editing, schedule validating, and automatically resolving conflicts.

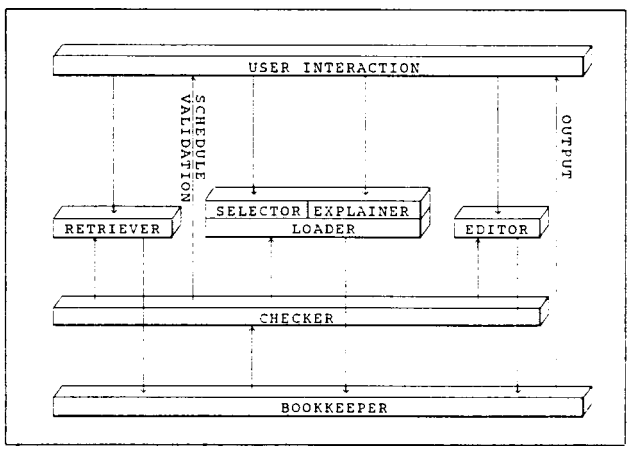


Figure 1. Major Components & Interfaces

THE BOOKKEEPER

Assume that a mission planner wants to schedule some activities which require power. Also assume the planning period is one week and the available power is 10.0 kilowatts. The planner would initialize a table like the one shown in Figure 2 with the initial time and value and the final time and a value of zero. The table is read: beginning at time 0/00:00:00 (zero days, zero hours, zero minutes, zero seconds), the power available is 10.0 kilowatts; at 7/00:00:00, the power available changes to 0.0 kilowatts.

Suppose that an activity is scheduled from 1/13:45:00 to 3/16:00:00 using 2.5 kilowatts of power. Also suppose that another activity is scheduled from 4/03:28:45 to 4/09:00:15 using 3.75 kilowatts of power. Four new times are inserted into the table showing the reduction and restoration of power by each activity.

The process becomes more intricate when activities overlap each other. Assume that another activity is added from 3/16:00:00 to 5/18:35:30 using 1.25 kilowatts of power. Since the start time already exists in the table, it is only necessary to insert one new time. However, the power availability for all the time points between the start and stop times must be updated.

Initial		With 2 inserts		With 3rd insert	
Time	Power	Time	Power	Time	Power
0/00:00:00	10.00	0/00:00:00	10.00	0/00:00:00	10.00
7/00:00:00	0.00	1/13:45:00	7.50	1/13:45:00	7.50
		3/16:00:00	10.00	3/16:00:00	8.75
		4/03:28:45	6.25	4/03:28:45	5.00
		4/09:00:15	10.00	4/09:00:15	8.75
		7/00:00:00	0.00	5/18:35:30	10.00
				7/00:00:00	0.00

Figure 2. Sample Bookkeeper Table

This example is lacking only in scale! Doing the task manually would require keeping up with dozens of resources and thousands of events. In addition to keeping up with resource and crew usage, it is also necessary to keep up with the activity schedule so that sequencing, concurrency, and performance delays can be checked, and to keep up with crew location so that crew translation time can be allotted. Notice that the length of the table is a function of the number of resource level changes, not the duration of the flight increment being scheduled.

The bookkeeper in ESP2 emulates the manual process described above by using specially designed file formats and processing techniques to rapidly access and update the data. Updating all the intervening time points between the start and stop times of an activity consumes time but permits the checker to operate much more efficiently. Since ESP2 is implemented on a 32-bit machine and time is maintained in integral seconds, the flight increment length is limited to the largest integer that can be stored in a 32-bit word; i.e., 2,147,483,654 seconds or 68+ years.

Space Station payload activity scheduling philosophy calls for scheduling models or groups of models within resource allocation envelopes. In this case, the bookkeeper is pre-loaded from a database containing the envelopes.

THE CHECKER

The data from the bookkeeper and the model requirements can be used to derive availability windows for most model requirements. A mission planner performing the checking function might want to know, "Where does the available power first exceed 8.00 kilowatts?" A quick scan of the table in Figure 2 would determine that the window opens at 0/00:00:00 and closes at 1/13:45:00.

If the next window were needed, the mission planner would remember how far down the table checking had proceeded, resume there and find that the next window runs from 3/16:00:00 to 4/03:28:45, followed by the window from 4/09:00:15 to 7/00:00:00.

In ESP2, the request presented to the checker might be, "Give me the first window where all the resources required for my activity are available." The request would also contain earliest and latest times of interest. This limiting of the search range allows the checker to respond much faster. The rules for converting the model requirements to windows reside within the checker. It can respond to questions about availabilities of orbit opportunities, equipment, nondepletable resources, or crew. It can also respond to queries about the windows where performance delay, sequencing, and concurrency requirements are met. Allowing time for crew movement from one location to another is accomplished by subtracting the translation time from each end of the crew availability windows. A matrix of translation times is provided in the database.

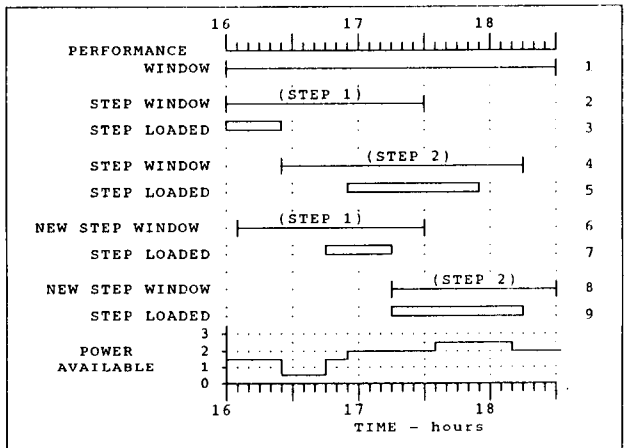
THE LOADER

The first step in describing how the loader works is to describe how a mission planner would load a single step into the schedule. After checking to see that the consumables were available, the mission planner would determine the window in which the step might be scheduled. The planner would then choose one of the requirements and find the first window satisfying that requirement. If this window were long enough, the mission planner would check for the window of another requirement. Checking would continue in this manner until all requirements were met. If the intersection of the windows were as long as the step, the step could be scheduled. In ESP2 the loading, or scheduling, of steps is accomplished in a manner similar to the human process described above. The requirements are checked in a particular order; i.e., the windows have a defined hierarchy. Each window is the search range for determining the window below it in the hierarchy. Whenever a window is unacceptable (shorter than the minimum step duration), the next window in the same search range (next higher window) is requested. Whenever no usable window is found at a particular level, the window above it becomes unacceptable and the next window at that level is checked. Checking continues in this manner until a set of nested windows for all requirements has been generated or there are no more windows at the highest level and the step cannot be loaded. The reader has probably noticed that the bulk of the summary given in a preceding section was merely a synopsis of the loader.

Once a lowest-level acceptable window is found, the step is loaded within this window according to several rules. The highest priority rule is to start the step as early as possible (front load). Another rule is to maximize the step duration within the window up to the limit specified on the model. If after assigning the start and stop times of the step there remains a choice of crew members to assign, then they are assigned so as to balance the crew usage.

But the task is not just to schedule steps, but to schedule performances of multi-step models! This task requires finding a place in the schedule which meets not only the requirements of all steps individually, but also the delay constraints between steps and performances, the resource carry-through requirements, the sequencing and concurrency requirements, and the performance windows specified with the model.

Confronted with this larger task, the mission planner would begin by defining a window in which the performance must be scheduled. This window is determined by considering the window from the selector, the performance window from the model, performance delays, and sequencing. The mission planner would load the steps in this window on a trial basis, moving them around until all steps were validly loaded. Only then would the bookkeeping function be conducted and the performance actually scheduled. A detailed example of front loading a performance is given in Figure 3.



MODEL:

Performance window: 0/16:00 to 0/18:30

Step 1:	Operate air sample collector	
Duration:	0/00:20 to 0/00:30	POWER: 0.889
Step 2:	Analyze air sample	
Delay:	0/00:00 to 0/00:15	
Duration:	0/01:00 to 0/01:00	POWER: 1.762

The program computes the window for the first step to be loaded (item 2 above), beginning at the start of the performance window (item 1) and leaving enough room for the remaining steps. The step is then loaded as early as possible for as long as possible within the window (item 3). The window for the second step (item 4) is similar to the first step except that it starts after the end of the first step by an amount equal to the minimum step delay. The loading of the second step (item 5) is delayed by resource availabilities. Note that the step delay relative to step 1 is violated. The window for step 1 is recomputed (item 6) and the step is again loaded (item 7) as early as possible for as long as possible. Note that the step overlaps step 2. The window for step 2 is recomputed (item 8), and the step is reloaded (item 9).

Figure 3. Performance Loading with Backtracking

Model requirements such as the delay between steps, resource carry-through, crew lockin, maximum performance duration, and others are implemented as backtracking rules in the loader. Firing one of these rules results in already-loaded steps being reloaded. When reloading steps, the program adjusts the step window so that the same backtracking rule is not violated again.

The above discussion does not take into consideration all of the requirements that exist in the domain specified for ESP2. Other rules are included to check for consumables. If consumable usage is a function of time, the step durations may have to be chosen at less than the otherwise available maximum. The scheduling and descheduling of startup and shutdown steps must be performed and crew monitoring must be scheduled.

After all steps of a performance are loaded, the performance is scheduled and all bookkeeping functions are conducted. In addition to asking the bookkeeper to update its data, the loader also updates the crew balancing parameters, the grading equation parameters and other data.

ESP2 does one more thing that a good mission planner would do. After scheduling a performance of a model, the program saves a snapshot of its computed data. When asked to schedule another performance of that model, the program resumes scheduling based on the snapshot. This heuristic feature significantly enhances the response time. However, it is possible for a snapshot to be invalidated by the scheduling of another model. Therefore after scheduling each performance, ESP2 checks all snapshots and clears those that are invalid.

This is only half of the story! When scheduling models with two-way concurrency (neither model can be scheduled without the other), ESP2 processes them simultaneously. The complexity of the loading task is at least doubled. Since both models are active at the same time and may require the same resources, the checker must account for the already loaded steps of the other model when computing availability windows. Determining when and how to backtrack is more complex and the computation of step windows is more difficult.

The loader has a last-chance ploy that is invoked after all other attempts to schedule a performance have failed. If any steps have variable durations, these durations are forced to the minimum value and the loading process is repeated. Remember, the original desire was to maximize step durations.

The preceding description applies to front loading; i.e., loading the performance as early as possible. Back loading (loading the performance as late as possible) is the mirror image of front loading.

THE SELECTOR

The loader places the models in the schedule based on model requirements and current availabilities. Since the program cannot deschedule or reschedule a performance (global backtracking) while generating a schedule, the order of attempting to schedule the models, referred to as the selection order, has a significant effect on the schedule. Selecting the next model/performance to schedule is the function of the selector. The selector also determines the topmost window in the performance-level hierarchy, the steps to be scheduled (the model scenario), and the loading algorithm.

The selector may specify the topmost window so as to force the loader to place the performance as required by a particular scheduling strategy. Possible scheduling strategies are resource usage leveling, reserving certain times for other yet to be scheduled activities, etc.

As part of the input database, the user specifies multiple scenarios for executing a performance of a model and the value, or weight, of each. Normally the selector requests that the loader attempt to schedule the highest-valued scenario; and, if it fails, the selector requests the next lower scenario; and so forth. Alternately, the user can specify a selection method that selects an equal number of each scenario or selects them proportional to the values.

The user also specifies which loading algorithm to use or specifies that the selector itself is to choose the algorithm. But most importantly, the selector chooses which model to schedule next. The user divides the models to be scheduled into groups, assigns a selection method to each group, and then specifies the order in which to process the groups. The selector processes the groups and passes the models to the loader one performance at a time. The four most commonly used selection methods are described below.

The fixed-order method allows the user to specify the complete order of selecting the models, possibly on a performance-by-performance basis. The user also specifies the rules for selecting the model scenario. When processing a fixed-order group, ESP2 makes automatic adjustments to account for sequencing and concurrency. ESP2 supplies a command which can reorder the members of a fixed-order group so that the most difficult to schedule is selected first. This command considers the time windows, the orbit opportunity requirements, and the number and duration of performances requested by the model.

The random-order method requires the user to specify a seed for a pseudo-random number generator, the group members, relative weighting factors for the members, and the scenario selection rules. As it processes a random-order group, ESP2 takes into account sequencing and concurrency. At the user's request, ESP2 automatically generates

multiple schedules and saves the "best" one. This Monte-Carlo technique allows the program to perform exhaustive searches without further user intervention.

The maximize-grade method allows the user to specify the members of the groups, the weighting factors for each of the schedule grading parameters, and special scenario selection rules. The scheduling order is determined by the selector based on which model (if scheduled) would yield the maximum increase in the grade.

The program also provides a pseudo or manual selection method. The scheduler editor provides a gateway to the automatic scheduler which bypasses the selector. In this mode the user performs the task of the selector — selecting the model and specifying the scenario and the topmost window.

THE EXPLAINER

As a companion to the loader, ESP2 provides a package which traces the step-by-step activities of the loader. This package can help the user understand why the program could not schedule another performance of a model or why it scheduled the performance where it did. The program usually cannot tell why a model cannot be scheduled. Indeed, there is often not a single reason. The crew may be available when the orbit opportunity is not, or the orbit opportunity may be available when the power is not, etc. No one reason can be stated for the failure. The trace package can show (literally) why the performance could not be scheduled. In order to interpret the trace, the user only needs a basic understanding of the scheduling process.

Primarily, the trace shows the windows returned by the checker and the firing of backtracking rules. The data delivered by the selector and feasibility test failures are also shown. The text presentation contains messages like the following:

- "resource window from 17/23:45 to 18/3:30", — window returned from the checker,
- "end of window checking crew", — the checker could not find a window within the specified search range (next-higher window),
- "window unacceptable", — window is shorter than the minimum step duration,
- "step 4 scheduled from 18/1:25 to 18/2:25", — step is tentatively loaded,
- "delay from step 3 to step 4 violated", — the loader must backtrack and reschedule step 3
- "chosen crew: 1,3", — crew members 1 and 3 are assigned to the step, and
- "minimizing" — the loader has invoked the last-chance rules.

These are only a sampling of the 74 possible messages.

The trace display is divided into two windows; one window containing the text and the other the graphics. Only the messages containing times (window and scheduled messages) are plotted. Figure 4 shows a typical trace display for a model without two-way concurrence. When models with two-way concurrence are scheduled, both are loaded and traced at the same time. Then both sets of performance level windows are shown above the dotted line. Since the loader only processes one step at a time, only one is shown below the dotted line.

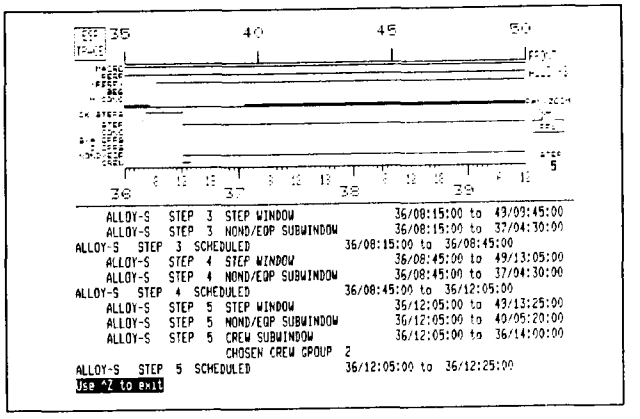


Figure 4. Typical Trace Display

The scale of the performance level portion of the plot is determined by the topmost window. The user may pan and zoom in the bottom (the step level) portion of the plot. The user may also scroll forward and backward through the messages. As each window or scheduled message is presented in the text window, it is plotted in the graphics window.

SUMMARY

In the Expert Scheduling Program all model requirements are reduced to feasibility tests, windows, or backtracking rules. Which model/performance is scheduled next is determined by the selector. The checker calculates the windows, and the loader combines the windows and executes the backtracking rules. The explainer presents the user with a trace of the loading process.

This technique has been proven over the last ten years by scheduling Spacelab payload activities. Only two conditions are necessary for it to support Space Station payload activity planning: the activity requirements must be reducible to a combination of feasibility tests, windows, or backtracking rules; and robust selection methods must be found to eliminate the now-required assignment of selection methods by an expert user.

Surveys of future payload descriptions have uncovered only a few requirements which are difficult to express in the required formats; for example: after-effects of an activity require including a step in the model which uses the affected resource.

The portfolio of selection methods in ESP2 is easy to expand and many additional scheduling strategies can be implemented as selection methods. Several additional selection methods have been designed and are being implemented. An expert system for dividing the payload activities into groups and choosing the best selection method (scheduling strategy) for each is being considered.

There is a parallel effort within our group and the community at-large to formulate a different (and better) method to schedule Space Station payload activities. As with any research effort, there is no guarantee that the research will be be successful, especially within the limited time span available before Space Station will become a reality.

ACKNOWLEDGEMENTS

The authors would like to acknowledge Mr. Jerry Weiler, the scheduling expert whose guidance was invaluable.

BIBLIOGRAPHY

1. Britt, D.L.; Geoffroy, A.L; Schaefer, P.R.; Gohring, J.R.: "Scheduling Spacecraft Operations", Conference on Artificial Intelligence for Space Applications, Huntsville, Al., Nov. 13-14, 1986.

2. Deuermeyer, B.L.; Shannon, R.E.; Underbrink, A.J.,Jr.: "Creation of the Selection List for the Experiment Scheduling Program (ESP)", Industrial Engineering Division, Texas Engineering Experiment Station, Texas A & M University, College Station, Texas. Final Report of NASA Contract No. NAS8-35972, NAS8-1778861, May, 1986.

3. Kurtzman, C.R.: "Time and Resource constrainted Scheduling, with Application to Space Station Planning", Ph.D. Thesis, Massachuetts Institute of Technology, Cambridge, Ma., 1988

4. Maxwell, T.G.: "Investigation of Experiment Selection Order Determination for Space Mission Scheduling", M.S. Thesis, University of Alabama in Huntsville, 1987.

5. Stacy, K.L.; Jaap, J.P.: "Space Station Payload Operations Scheduling with ESP2", Second Annual Workshop on Space Automation and Robotics (SOAR 88), Dayton, Ohio, July 20-23, 1988.